

UB-ANC: An Open Platform Testbed for Software-Defined Airborne Networking and Communications

Sayed.Jalil Modares Najafabadi, Nicholas Mastronarde
The State University of New York at Buffalo

Michael J. Medley, John D. Matyjas
Air Force Research Lab, Rome, NY

Abstract

We present the University at Buffalo's Software-Defined Airborne Networking and Communications Testbed (UB-ANC). UB-ANC is an open software/hardware platform that aims to facilitate rapid testing and repeatable comparative evaluation of various airborne networking and communications protocols at different layers of the network protocol stack. In particular, it combines quadcopters capable of autonomous flight with USRP E310 embedded software-defined radios (SDRs), which enable flexible deployment of novel communications and networking protocols, and real-time reconfigurable cognitive radio frequency (RF) links. This is in contrast to existing airborne network testbeds, which typically rely on inflexible off-the-shelf radios based on fixed standards and protocols, e.g., IEEE 802.11 (Wi-Fi) or IEEE 802.15.4 (Zigbee).

UB-ANC is designed with emphasis on modularity in terms of both hardware and software, which makes it highly customizable. It is built on two open-source frameworks, namely, GNU Radio for the physical and medium access control layers and C++ Qt for the higher layers of the network protocol stack including the network, transport, and application layers. In this paper, we present the hardware and software architectures underlying UB-ANC, describe the implementation and operation of its most important components, and discuss how these components can be modified to achieve desired behaviors.

1 Introduction

Airborne networking and communications (ANC) has emerged as an important technology for public safety, commercial, and military applications including search and rescue, disaster relief, environmental monitoring, and C3ISR (i.e., command and control, communications, intelligence, surveillance and reconnaissance). However, the design and implementation of airborne networks poses unique challenges due to their fluid topologies and limited node lifetimes, and the fact that they are *cyber-physical systems* [58]. In particular, both node mobility and limited node lifetimes may lead to intermittent/broken links or even a partitioned network.¹ While broken links can often be dealt with by rerouting traffic through other nodes, network partitioning must be mitigated through controlled mobility and data ferrying to maintain connectivity [53]. In other words, the underlying communications and networking capabilities (i.e., the cyber component of the airborne network) cannot be considered independently of the physical locations and flight-paths of the nodes, and vice versa [58].

The synergy between the cyber and physical components of an airborne network becomes even more important in mission-critical C3ISR applications, where the communication network must simultaneously support the distribution of command and con-

¹A network is said to be partitioned if there does not exist a direct link, nor a multi-hop path, from one set of nodes to another.

trol, intelligence, surveillance, and reconnaissance information, while operating in a potentially adversarial network environment characterized by congested and contested spectrum and low and intermittent bandwidth. Within this challenging environment, many techniques can be leveraged at the physical, data link, network, and transport layers to improve a variety of cyber metrics such as throughput, communication reliability, latency, energy-efficiency, and/or spectral efficiency (see, e.g., [49, 50, 54, 61, 62]). However, these metrics alone cannot tell us how well the underlying networking and communications capabilities support C3ISR applications because they do not tell us anything about the physical behavior of the airborne nodes (whether or not they accomplish their mission, if they accomplish it in a timely manner, etc.).

Although the physical characteristics of an airborne network can be simulated [57, 58, 60, 65], actual implementations and field-tests have been recognized as crucial for demonstrating and evaluating solutions in real-world operating environments [47, 48, 53]. To this end, we have developed a software-defined ANC testbed at the University at Buffalo (UB). The testbed, which we call UB-ANC, combines quadcopters that are capable of autonomous flight with software-defined radios (SDRs) that enable flexible deployment of novel communications and networking protocols on the quadcopters. In particular, UB-ANC provides us the ability to collect data to measure and understand the connection between a multi-agent ANC system's underlying networking and communications capabilities and the ability of the agents to effectively accomplish a given mission in a given network environment.

There has been a lot of interest in the research community on airborne nodes/drones equipped with networking and communications capabilities to support different applications. In [60], Rohrer et al. develop a domain-specific architecture and protocol suite for cross-layer optimization of airborne networks. They introduce a TCP-friendly transport protocol, IP-compatible network layer, and geolocation aware routing. They perform simulations of their protocols in network simulator programs (ns-3 and ns-2), but do not verify their performance in a

real system. In [65], the authors propose the Mobility Aware Routing / Mobility Dissemination Protocol (MARP/MDP) to reduce latency and routing overhead by exploiting the known trajectories of airborne nodes. The nodes' trajectories are preplanned to maximize network connectivity using techniques in [64]. MARP/MDP is compared against OLSR and AODV using the QualNet network simulator. In [57], Le et al. simulate a reliable user datagram protocol in OPNET Modeler v14.5 and in [58], Namuduri et al. discuss cyber-physical aspects of airborne networks and use ns-2 to study average path durations under different node velocities, hop counts, and node densities.

In [56], researchers in the University of Pennsylvania's General Robotics, Automation, Sensing, and Perception (GRASP) laboratory discuss autonomous navigation of unmanned air vehicles, including swarms of quadcopters that fly in complex formations. This system relies on external localization, centralized planning, and an ideal RF environment to enable perfect coordination among quadcopters in the swarm. Note that off-the-shelf 802.15.4-compliant Zigbee radios are used to enable communication between a centralized control node and the group of vehicles under its control. In [52], researchers discuss the Group Autonomy for Mobile Systems project at Carnegie Mellon University, which focuses on developing interfaces and artificial intelligence algorithms to enable a single operator to control multiple unmanned systems. This work relies on a network to enable airborne nodes to collaboratively cover a rectangular area, but does not consider an adversarial network environment. In [66], Zhang proposes a minimalist swarming sensor network of expendable nodes, where each node is designed to tolerate and recover from physical failures (e.g., crashes). While the swarm of nodes is controlled in a distributed way over a communication network, this work does not consider operation of the system within an adversarial network environment.

In [47], Allred et al. study airborne wireless sensor networks for atmospheric, wildlife, and ecological monitoring. They equip airborne nodes with off-the-shelf 802.15.4-compliant Zigbee radios. They perform experiments to evaluate the performance of air-to-air,

air-to-ground, and ground-to-ground wireless links, as well as network connectivity. In [53], researchers at the University of Colorado test the performance of off-the-shelf IEEE 802.11b (Wi-Fi) networking equipment in an airborne mesh network. They show that a mesh network can extend the communication range among airborne nodes in a small unmanned aerial system (UAS), they explore how a mesh network can be used to enable a remote operator to send command and control information to distant aircraft and to receive telemetry information back over the network, and they use controlled mobility to enable ferrying of delay-tolerant data between nodes in a fractured/partitioned network.

Only recently have researchers started to study the benefits of equipping drones with SDRs [51, 55, 67]. In [51], dos Santos et al. design and implement a drone equipped with a low-cost SDR receiver, which can automatically track wildlife tagged with very high frequency (VHF) radio collars. In [55], Jakubiak implements a drone equipped with a low-cost SDR receiver to gather data about the coverage of a cellular network. In [67], Zhou envisions an Unmanned Aircraft System for Railways in which drones will relay data for passengers in high-speed trains to different networks (e.g., satellite or cellular). While [51, 55] implement systems with only SDR receivers, [67] does not provide any system implementation.

In summary, while a lot of significant contributions have been made in designing and implementing multi-agent airborne networks, which exploit communications and networking technologies for command and control, telemetry, and coordination among multiple agents, existing system implementations typically assume a favorable (near lossless) communication environment and either rely on inflexible off-the-shelf transceivers or SDR-based receivers. In contrast, UB-ANC provides a flexible and highly reconfigurable multi-agent ANC platform for testing state-of-the-art communications and networking protocols implemented on SDR transceivers. This affords us the opportunity to not only study how these advanced capabilities can harden the airborne network in adversarial network environments, but to also evaluate how well the airborne nodes can accomplish different missions in such environments.

In this paper, we describe the design and implementation of a modular, general purpose, open platform with reconfigurable communications and networking capabilities, which can be easily modified for rapidly testing and benchmarking various algorithms at different layers of the network protocol stack. We make the following contributions:

- We identify a set of core components of a drone agent that are needed for a general purpose platform.
- We define a modular architecture, in terms of hardware and software, to achieve maximum customizability.
- We present an implementation of our architecture using Qt, GNU Radio, and the USRP E310 SDR embedded platform.
- Finally, we explain how to make appropriate modifications to the system components to achieve desired functionality of the platform.

The rest of the paper is organized as follows. In Section 3, we discuss UB-ANC's hardware design and its components. In Section 4, we explore UB-ANC's software architecture, its main modules, and provide details about each module's implementation. In Section 5, we explain how to make appropriate modifications to the system to achieve desired functionality for different applications. In Section 6, we describe our planned future work based on UB-ANC. We conclude the paper in Section 7. UB-ANC software is available for public use as described in Section 9.

2 Core Requirements of a Drone Agent

In general, a drone agent needs to have a high performance CPU (and preferably FPGA) to run advanced algorithms for communications, networking, and mission planning; an efficient radio frequency (RF) system to communicate with other agents or ground stations; a robust autopilot system to control the drone's movements; a sufficient power system

to support the weight of the payload (i.e., the communications system and any sensors) and to achieve satisfactory flight times; and appropriate sensors for gathering information from the environment.

Up to now, we have designed and implemented two generations of the UB-ANC drone. The 1st generation UB-ANC drone used a Matchstiq SDR [20] and a Turnigy Talon Quadcopter V2.0 Carbon Fiber frame [39], which we modified with some 3D printed components to carry the SDR and a separate battery pack to power the SDR. Unfortunately, the 1st generation quadcopter prototype had an inadequate flight time, lacked support for GNU Radio, lacked support for additional components/sensors (because the frame could not easily accommodate more components, the propulsion system could not operate with a heavier drone, and there were insufficient interfaces on the Matchstiq SDR), and was very fragile due to the 3D printed components.

To address these limitations, we have created a 2nd generation quadcopter prototype, which includes a new custom-built quadcopter frame and a USRP E310 embedded series SDR from Ettus Research [41]. The USRP E310 is an open hardware [42], and has additional interfaces to support additional components/sensors. Table 1 shows a brief comparison between the two quadcopter prototype generations. In the following sections, we explain in detail important components of the hardware, software, and quadcopter frame designs.

3 Hardware Components

There are three main components to UB-ANC's hardware architecture design, namely, the Central Control Unit (CCU) to control the agent's behavior, the SDR to communicate with other agents, and the autopilot controller for flight. The current version of UB-ANC uses the USRP E310 for both the CCU and SDR, and uses the 3D Robotics ArduPilot Mega (APM) 2.6 [5] for the autopilot controller. The USRP E310 is connected to the APM 2.6 through a USB interface with APM 2.6 as a device and USRP E310 as the host. Figure 1 shows an overall diagram of the hardware design.

3.1 Central Control Unit (CCU) and Software-Defined Radio (SDR)

The USRP E310 includes a Xilinx Zynq-7020 All Programmable SoC [46] and is used as the CCU and SDR. In terms of CCU, it has a low power, high performance dual-core ARM Cortex-A9 processor, running at 667 MHz. In terms of SDR, it has an Artix-7 FPGA [10] and AD9361 RFIC [2], which provides 2x2 MIMO support covering 70 MHz — 6 GHz and up to 56 MHz of instantaneous bandwidth.

One of the most important aspects of the Zynq architecture is that the CPU and FPGA are on the same chip, connected through a high performance ARM AMBA 4 AXI bus [3]. This eliminates bus delays, which some SDRs suffer from because the CCU and SDR are connected via USB or Ethernet [59]. RF Network on Chip (RFNoC) [38] is another advanced technology used in the 3rd generation USRP SDRs, which dramatically simplifies the development of high-performance SDR applications running on both the FPGA and embedded CPU. For processing, the USRP E310 has 1 GB DDR3 RAM for the ARM CPUs and 512 MB DDR3 RAM for the FPGA logic. It also has interfaces for 10/100/1000 BASE-T Ethernet and host USB support, and a microSD card slot for storing the kernel, FPGA bitstream, system file images, and data logs.

3.2 Autopilot

As we mentioned before, each UB-ANC drone is equipped with an APM 2.6 autopilot control system, which is based on the Arduino open platform project [8]. APM 2.6 enables the quadcopter to be flown in either a programmed autonomous mode or a manual control mode. To support autonomous flight, it includes numerous sensors such as a 3-axis gyro, accelerometer, magnetometer, barometer, and GPS. It uses Atmel's ATMEGA2560 [11] for processing, and it uses ATMEGA32U-2 for USB functions. It also features onboard 4 MB dataflash chip for automatic data-logging.

An external 3D Robotics u-blox GPS with Compass [1] is used to provide high GPS accuracy for the autopilot controller. It features active circuitry

Table 1: Comparison between two UB-ANC drone generations

	1st Generation	2nd Generation
Flight Time	less than 7 minutes	more than 25 minutes
Construction	Fragile	Sturdy
Open Hardware	No	Yes
GNU Radio Support	No	Yes
Additional Component/Sensor Support	No	Yes

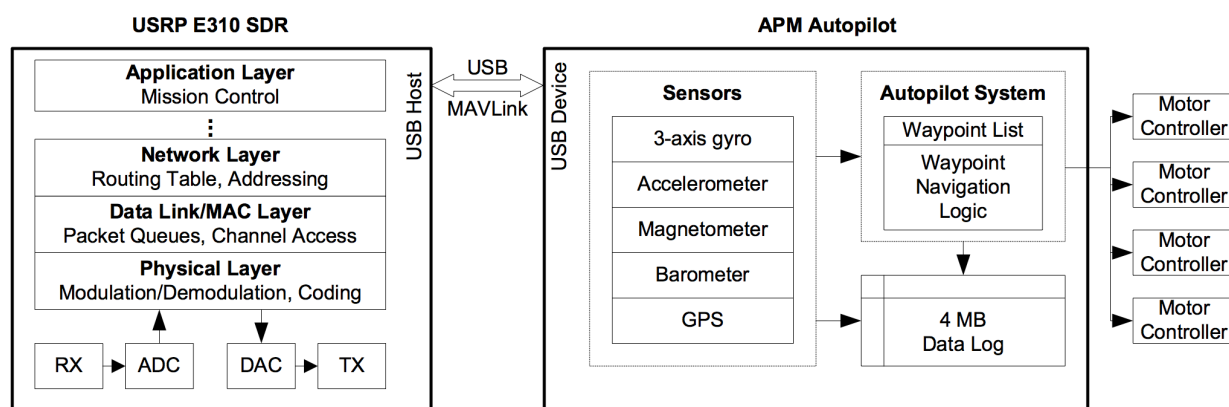


Figure 1: Block diagram illustrating how the USRP E310 SDR (USB Host) and APM 2.6 Autopilot Controller (USB Device) are connected via a USB

for the ceramic patch antenna, rechargeable backup battery for warm starts, and an I2C EEPROM for configuration storage.

3.3 Quadcopter Platform

Although the focus of our research is not on the airframe, propulsion, or control system, we have invested considerable time to ensure that the airframe can physically accommodate the payload (i.e., the SDR), that the propulsion system comprising the battery, motors, and propellers can provide the necessary thrust and flight times for experimentation, and that the commercial off-the-shelf control system supports stable flight, appropriate fail-safes, GPS waypoint navigation, and USB connectivity for interfacing

with the software-defined radio.

The quadcopter itself is custom-built with a carbon fiber frame, aluminum landing gears, and alloy steel nuts and bolts. The frame has been designed to accommodate components with different sizes and weights (e.g., different batteries, SDRs, autopilots, and sensors) and therefore is extensible from its current configuration. Without the flight battery and payload, the quadcopter weighs approximately 1.55 kgs; with the flight battery (a 10,000 mah 22.2 V 6-cell LiPo) and USRP E310 SDR, the weight increases to approximately 3.125 kgs. The USRP E310 itself weighs 375 grams when equipped with a full enclosure; however, a partially enclosed version is available that only weighs 225 grams.

The quadcopter's propulsion system (comprising

the selected battery, motors, and propellers) can provide a total of 9.6 kgs of thrust, which is more than adequate for the fully loaded quadcopter.² Indeed, through preliminary testing, we have achieved excellent flight times: 26 minutes with the USRP E310 SDR and 37 minutes without.

In Section 9, we provide a link where the interested reader can find more detailed information about the frame, factor of safety analysis [13], motors, propellers, and bill of materials (BOM). A 3D rendering of our quadcopter platform is shown in Figure 2. For clarity, the RC receiver, motor speed controllers, USRP E310 SDR battery, and wires are not illustrated. Figure 3 shows one of our two fully implemented UB-ANC drones.

We note that our implemented software and hardware designs are not restricted to our quadcopter platform. For example, we could instead mount the USRP E310 SDR on the DJI Matrice 100 [21] customizable and programmable flight platform; however, it costs approximately 3x as much as our custom-built platform (excluding the cost of the SDR).

4 Software Components

In the previous section, we described the main hardware components of UB-ANC. In this section, we examine important components of the software architecture. The USRP E310 runs Yocto Linux as its operating system [45] and the APM 2.6 runs ArduPilot APM:Copter [7] as its firmware. The Yocto Linux and APM:Copter are connected to each other using USB CDC-ACM [40] as a serial port with baud rate 115200 bps.

In the software architecture design, there are four main components, namely, the Autopilot Control Unit (ACU), Network, Logging Unit (LU), and Agent. Figure 4 provides a high-level diagram of the software architecture.

The upper layers of the network protocol stack (the

²An informal rule-of-thumb is that the propulsion system should provide at least twice as much thrust as the weight of the quadcopter; therefore, our design can support a heavier payload than our SDR alone.

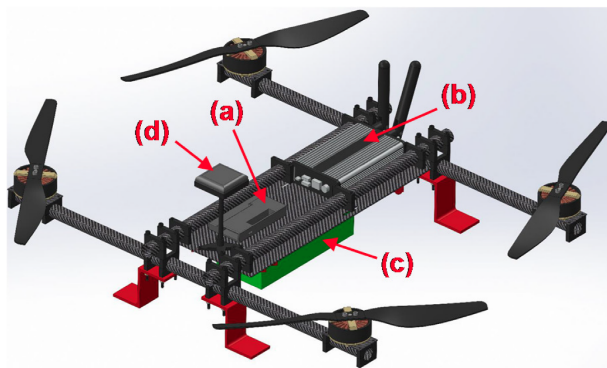


Figure 2: 3D rendering of our quadcopter platform (RC receiver, motor speed controllers, SDR battery, and wires are not illustrated). (a) APM 2.6 Autopilot Controller; (b) USRP E310 SDR; (c) Flight system battery; (d) GPS module

network layer through the application layer) are implemented using Qt [35], which is an object-oriented C++ cross-platform application development framework. The Qt Signal/Slot mechanism is used to enable various software components to communicate with each other. We have chosen Qt as the main application framework based on the following considerations:

- It facilitates event-driven programming. Specifically, every task / component can be implemented as an object that can easily communicate with other objects using Qt's Signal/Slot mechanism.
- It is a stable open-source application framework that has been used in many other open-source projects. In particular, some of the open-source software that we are using in this project is already implemented using Qt. Thus, Qt allows us to reuse this software so that we do not have to reinvent the wheel.
- It is a C++ object-oriented framework, which facilitates efficient coding and modularity, while also maintaining high-performance operation for implementing mission-critical applications.

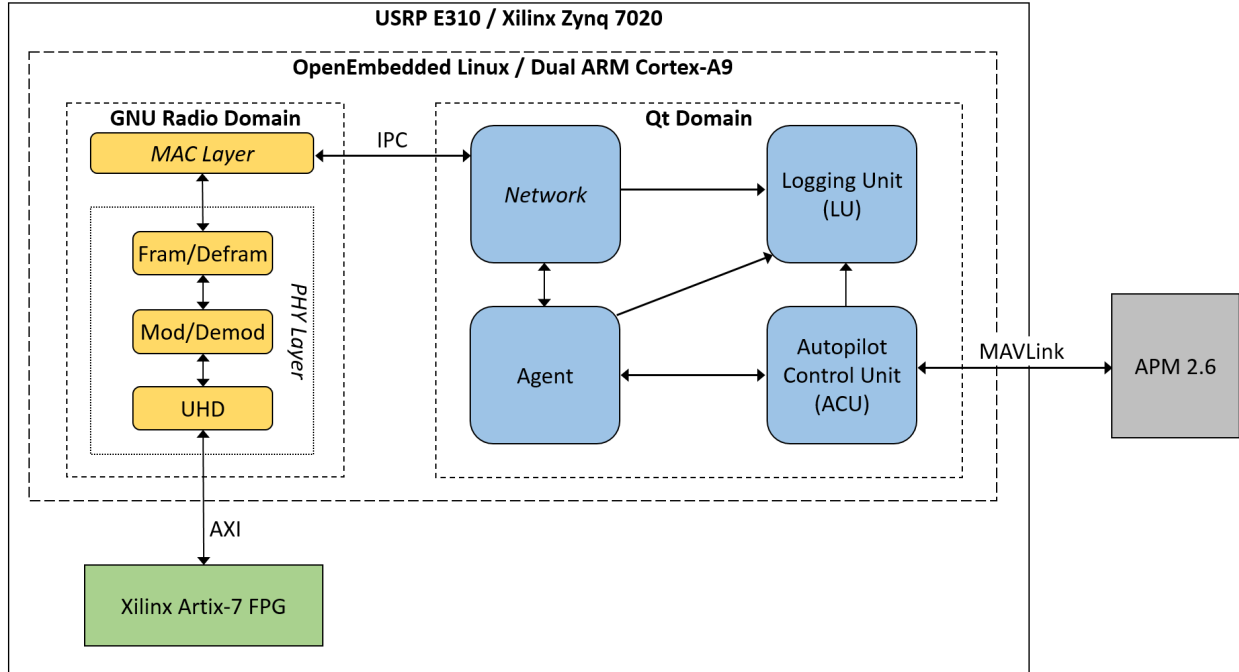


Figure 4: High-level software architecture diagram

- It is cross-platform, which makes it easy to port the project across different Operating Systems (OS), like Windows CE, Custom Embedded Linux, Android, and iOS.

The lower layers of the network protocol stack (i.e., the MAC and PHY layers) are implemented using GNU Radio [16], which is an open-source software toolkit that provides a digital signal processing execution environment for processing on a host computer or embedded CPU. For higher performance, GNU Radio uses the Vector-Optimized Library of Kernels (VOLK) [44] to utilize the processor's SIMD instructions (MMX, SSE, SSE2, SSE3, and SSE4 for Intel processors and NEON SIMD for ARM Cortex-A series [27]). The GNU Radio and Qt domains are connected via an inter-process communication (IPC [19]) mechanism.

4.1 Autopilot Control Unit (ACU)

Overall, the ACU is responsible for controlling the behavior of the autopilot controller. The agent software component uses the ACU to send commands to the autopilot controller. In order to send/receive commands (or messages) to/from the autopilot controller, the ACU uses the MAVLink messaging protocol [25], which is a lightweight, header-only C library for micro air vehicles (MAV). Figure 5 shows the MAVLink frame structure, and Table 2 shows the description of each field.

MAVLink supports various messages and commands. One of the most important messages, called the HEARTBEAT message [17], is generated by the ACU and autopilot controller every second (1 Hz). The HEARTBEAT message shows that the link between the ACU and autopilot controller is still alive. If the HEARTBEAT message from the ACU is lost, then the autopilot controller goes into a failsafe mode,



Figure 3: The fully implemented UB-ANC drone

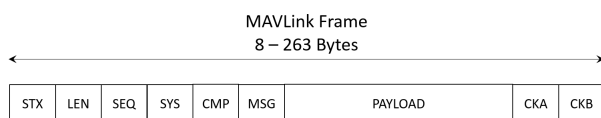


Figure 5: MAVLink frame structure

which means it will return to launch (RTL) if it has GPS lock or land if it does not. The HEARTBEAT message from the autopilot controller contains information that the ACU can use for different tasks. The information includes, but is not limited to, the following:

- **TYPE:** The type of micro air vehicle (quadcopter, helicopter, fixed wing, etc.)
- **AUTOPILOT:** The type of autopilot controller (APM, Pixhawk, etc.)
- **MODE:** The mode of the autopilot (armed, autonomous, manual, stabilize, etc.)
- **MAVLINK VERSION:** The MAVLink protocol version (the current version is 1.0)

Notice that not all MAVLink commands [22] are necessarily supported by a specific autopilot controller. For instance, the APM 2.6 only supports a

Table 2: Field description of the MAVLink frame

Byte Index	Description	Range
0	Frame Start Sign	0xFE
1	Frame Length	0 – 255
2	Frame Sequence	0 – 255
3	System ID	1 – 255
4	Component ID	0 – 255
5	Message ID	0 – 255
6 – (n+6)	Payload	0 – 255 Bytes
(n+7) – (n+8)	Checksum	SAE AS-4 Hash

Table 4: Parameters for loiter command

Param No.	Description
1	Seconds (decimal)
2	Empty
3	Radius around the waypoint, in meters
4	Desired yaw angle
5	Latitude
6	Longitude
7	Altitude

subset of MAVLink commands [9]. Table 3 shows some important MAVLink commands. Note that every command is associated with some parameters. For example, Table 4 shows the different parameters needed for the loiter time command.

Most of the classes in the ACU come from an open-source project called APM Planner 2 [6]. APM Planner 2 is a GUI based software, which can be used to define missions, send missions to the autopilot controller, and track the drone on a map. It is based on a project called Ground Control Station (GCS) [28]. APM Planner 2 and GCS are based on Qt and work with autopilot controllers compatible with MAVLink. Two important classes in the ACU are **LinkManager** and **UASManager**.

LinkManager: The **LinkManager** class is responsible for managing different kinds of links between the autopilot controller and the ACU (serial link,

Table 3: Some important MAVLink commands

CMD ID	Command Name	Description
16	MAV_CMD_NAV_WAYPOINT	Navigate to a waypoint
19	MAV_CMD_NAV_LOITER_TIME	Loiter around a waypoint for X seconds
20	MAV_CMD_NAV_RETURN_TO_LAUNCH	Return to launch location
21	MAV_CMD_NAV_LAND	Land at location
22	MAV_CMD_NAV_TAKEOFF	Takeoff from ground
176	MAV_CMD_DO_SET_MODE	Set system mode
183	MAV_CMD_DO_SET_SERVO	Set a servo to a desired PWM value

TCP/UDP link, telemetry link, etc.). Every link has a corresponding class (`SerialLinkInterface`, `TCPLink`, `UDPLink`, etc.), and when a link is established, the `LinkManager` creates the corresponding link object. The agent object then uses that link object to control the corresponding link (connect, disconnect, set baud rate, etc.). The following code shows how the agent object uses the `LinkManager` to check the type of link, and then sets some properties of the link (`SerialLinkInterface`) object.

```
void UBAgent::handleNewLink(int linkid) {
    if (LinkManager::instance()->
        getLinkType(linkid) ==
        LinkInterface::SERIAL_LINK) {
        SerialLinkInterface* link =
            static_cast<SerialLinkInterface*>
            (LinkManager::instance()->
             getLink(linkid));
        link->setPortName("ttyACMO");
        link->setBaudRate(BAUD_RATE);
        LinkManager::instance()->
            connectLink(linkid);
        connect(UASManager::instance(),
            SIGNAL(activeUASSet(UASInterface*)),
            this,
            SLOT(handleActiveUAV(UASInterface*)));
    }
}
```

UASManager: The `UASManager` class is responsible for managing different kinds of autopilot controllers (ArduPilot Mega, Pixhawk, etc.). When the ACU receives a HEARTBEAT message, the `UASManager` first determines the type of autopilot

controller that sent the message. If the corresponding autopilot controller's object does not already exist, then the `UASManager` creates the corresponding autopilot controller object (`ArduPilotMegaMAV`, `PxQuadMAV`, etc.) and puts it in its own private property called `m_uas_list`. The agent object then uses the autopilot controller object to send/receive any commands that are supported by the corresponding autopilot controller. The following code shows how the agent object uses the `UASManager` to get the corresponding (`ArduPilotMegaMAV`) object, and how it uses the `ArduPilotMegaMAV` object to send commands (it also uses the `UBNetwork` object to send data to other agents, which we will explain in more detail in Section 4.2). As can be seen in the code, `UBAgent` uses `executeCommand` method from the `UAS` class to send a command to the autopilot controller. `executeCommand` needs the following ten parameters: the type of command, the autopilot ID, the subsystem component ID, and the seven parameters needed by the specific command (see, e.g., Table 4 for the seven parameters required by the loiter command).

```
void UBAgent::handleActiveUAV(UAS* uav) {
    m_uav = static_cast<ArduPilotMegaMAV*>(uav);
    m_uav->setHeartbeatEnabled(true);
    connect(m_uav,
        SIGNAL(heartbeatTimeout(bool, uint)),
        this,
        SLOT(handleHeartbeatTimeout(bool, uint)));
    ...
}
...
void UBAgent::stageLoiter() {
```

```

if ((QGC::groundTimeSeconds() -
    m_loiter_timer > LOITER_TIME)) {
    m_uav->executeCommand(MAV_CMD_NAV_LAND,
        1, 0, 0, 0, 0, 0, 0, 0, 0);
    m_net->sendData(&m_msg);
    m_stage = STAGE_STOP;
    return;
}
...
}

```

4.2 Network

In order to communicate with other agents in the network, or with a ground station, the agent object uses the Network software component, which consist of three main parts, namely, the **UBNetwork** class, the MAC layer, and the PHY layer. As we mentioned before, the upper layers of the network protocol stack, including the application layer (**UBAgent**) and network layer (**UBNetwork**), are implemented in the Qt domain, while the lower layers (MAC and PHY) are implemented in the GNU Radio domain.

Note that nodes may exchange different types of information including intelligence, surveillance, and reconnaissance information, as well as command and control information. In other words, every node has the ability to send commands to other nodes in the network, e.g., GPS waypoints that the node should visit, a request to connect two regions of a partitioned network, or instructions to ferry data from one region of a partitioned network to another.

UBNetwork: The agent object uses the **UBNetwork** object to send/receive packets to/from other agents in the network. When a packet is sent to the **UBNetwork** object by the agent object, the **UBNetwork** object puts the packet into its own private property called **m_send_buffer** and then sends the packet to the MAC layer through an IPC mechanism, called protocol data unit (PDU [31]) socket. When a packet is received by **UBNetwork** from lower layers, it raises a signal and notifies the agent object that there is a packet in the **m_received_buffer** buffer. The agent object then reads the buffer and processes the received packet. The following code shows how the **UBNetwork** uses the Qt container classes to buffer

and unbuffer the data.

```

void UBNetwork::sendData(QByteArray* data) {
    QByteArray* stream = new QByteArray(*data);
    m_send_buffer.enqueue(stream);
}
...
QByteArray UBNetwork::getData() {
    QByteArray data;
    if (!m_receive_buffer.isEmpty()) {
        QByteArray* stream = \
            m_receive_buffer.dequeue();
        data = *stream;
        delete stream;
    }
    return data;
}

```

MAC and PHY Layers: The main functionality of the PHY layer is to receive/send data from/to the MAC layer, frame/deframe it [e.g., add a header and 32-bit cyclic redundancy check (CRC32)], modulate/demodulate it (using modulation schemes provided by GNU Radio, e.g., GMSK, DPSK, DPPSK, or even OFDM, or custom-built modulation schemes), and send/receive the frame to/from the USRP Hardware Driver (UHD) [43]. UHD is the hardware driver for all USRP devices. It is responsible for sending/receiving samples to/from the USRP radio system. The MAC layer controls the access to the RF channel, and can be programmed to support different MAC protocols, e.g., carrier sense multiple access with collision avoidance (CSMA/CA) or time-division multiple access (TDMA). The MAC and PHY layers are implemented as GNU Radio flow graphs [12] using GNU Radio Companion [15].

4.3 Agent

The Agent software component consists of **UBAgent** class, which is responsible for any mission that the drone is supposed to complete. The agent object continuously checks the state of the drone and the mission through a function called **missionTracker**, which is called every 10 milliseconds (100 Hz). Different kinds of multi-agent coordination techniques can be implemented with this object, like Multi-Agent Distributed Adaptive Resource Allocation

(MADARA) [26]. For advanced mathematical (matrix and vector) calculations the agent object uses `Alglib` [4], which is a cross-platform numerical analysis and data mining library. The following code shows a finite state machine algorithm for a simple take-off, loiter, and landing mission. As can be seen in the code, the agent object first checks if the autopilot controller is armed and then it goes through the different states.

```
void UBAgent::missionTracker() {
    if (!m_uav->isArmed()) {
        return;
    }
    switch (m_stage) {
        case STAGE_START:
            stageStart();
            break;
        case STAGE_LOITER:
            stageLoiter();
            break;
        case STAGE_STOP:
            stageStop();
            break;
    }
}
```

4.4 Logging Unit (LU)

There are lots of activities in the system that can be tracked. The tracked data includes, but is not limited to, GPS position (longitude, latitude, and altitude), MAVLink messages, drone speed, direction, Received Signal Strength Indicator (RSSI) [37], packet information (e.g., packet ID, size, source ID, destination ID, sender ID, receiver ID), frequency and gain of the RF system, etc. We track data in our system using `QsLog` [34], which is a system logger based on Qt's `QDebug` class. The data are logged on a MicroSD card so it can be analyzed offline, or it can be sent to a ground station where it can be viewed and analyzed in real-time.

5 Customization

One of the most important aspects of UB-ANC, besides the reconfigurable RF system, is that it can be

Table 5: Important classes and their responsibilities

Class Name	Responsibility
UBAgent	Agent behavior
UBNetwork	Network routing behavior
UAS	Autopilot controller behavior
LinkInterface	Link between ACU and autopilot

easily customized. In this section, we examine how different software components can be changed according to specific scenario. Table 5 illustrates some important classes along with their responsibilities.

ACU: As we explained earlier, the MAVLink protocol is used for communication between the ACU and the autopilot controller. While MAVLink has a common set of messages, custom messages can be created that replace or extend the common message set. There is a file named `minimal.xml` which contains a minimal set of MAVLink commands and messages. Then we can use `MAVLink Generator` [23] to generate the corresponding header files, which manipulate the custom messages. To handle custom messages in the ACU, we make a new class by inheriting the `UAS` class, and then implementing new versions of the class's methods [24]. With this approach, we can use different autopilot controllers such as the 3D Robotics Pixhawk [30], which uses Px4 Autopilot [32] as its firmware. If there is a need to change the media link between the ACU and the autopilot controller (i.e., serial, TCP/UDP, or telemetry), we should use `LinkInterface` as the base class, and then modify the corresponding class's method.

Agent: As we mentioned before, the drone agent behavior is controlled by the `UBAgent` class. Thus, for changing the drone behavior in the network, the `UBAgent` class should be taken as a base class, and the `missionTracker` method should be changed accordingly. The `enum TRACK_RATE`, which sets the frequency that the `missionTracker` function is called, can be set to a higher value for more critical missions or a lower value for less important missions.

Network: The functionality of the Network component can be modified by changing `UBNetwork` class.

We note that the network component contains both the network and transport layers of the protocol stack. Therefore, different routing and transport layer protocols can be implemented in this component.

MAC and PHY Layers: The lower layers are implemented as a GNU Radio flow graph. Thus, they can be replaced by other standard modules, including frame header generator, CRC32, Differential BSK (DBSK), DPSK, GMSK, OFDM, etc., or custom modules, called GNU Radio out of tree (OOT) modules [29]. There is a simple GNU Radio flow graph which can be used as a starting point of designing MAC and PHY layers.

Additional Components/Sensors: As we mentioned earlier, there are different interfaces on USRP E310, including two USB host ports and an Ethernet port, which can be used to connect additional components/sensors. Importantly, Yocto Linux supports a majority of interface drivers. In order to connect to the hardware, the Agent component software needs some custom functions, or classes to be added to the software design as a new software component. The custom classes can then use the corresponding driver to talk to the hardware. If a custom class needs to use Qt's signal/slot mechanism, it should use `QObject` [33] as its base class, and if it needs to run in separate thread, it should use `QThread` [36] as its base class.

6 Future Work

We will leverage the UB-ANC testbed for experimentation, testing, and benchmarking novel airborne networking and communications protocols. One major usage scenario that we envision for UB-ANC aims to capture essential elements of a mission. In this scenario, there are both cooperative and noncooperative nodes, which may be either airborne or located on the ground as illustrated in Figure 6. The cooperative nodes must collaboratively accomplish a set of dynamic waypoint navigation (or target assignment) tasks, which require frequent communication among the collaborative nodes (both air and ground) as well as intelligent, fully distributed, and autonomous com-

mand and control. Meanwhile, the noncooperative nodes create a congested or contested spectral environment, which forces the cooperative nodes to adopt sophisticated communications and networking strategies in order to successfully accomplish their tasks.

In order to evaluate the performance of different networking and communications technologies/protocols within UB-ANC, we will develop a set of benchmark test scenarios. Tests will be designed for indoor locations and outdoor locations (e.g., the Controlled Contested Environment at the Stockbridge Research Facility [18] and on campus at UB). A benchmark test scenario includes, but is not limited to, the following information:

Set of nodes: A benchmark test scenario has a specified number of adversarial, friendly, airborne, and ground nodes.

Starting positions: A benchmark test scenario specifies the initial positions of each node.

Leaders and followers: A benchmark test scenario defines roles for each node. Each node can be assigned as a leader or follower, where leaders serve as sources for command and control information, and sinks for intelligence, surveillance, and reconnaissance information. It is also possible that nodes can dynamically change roles over time.

Mission waypoints, timing, and information collection and dissemination: A benchmark test defines waypoints/targets for each airborne node, the times by which each node must reach each target, what information must be acquired at each target (e.g., audio, image, video), and where this information should be sent.

Prioritization and timing: All packets that are sent through the network must be tagged with priorities and timing information. Timing information defines the time by which the data must be delivered to its destination for it to be useful. The priority indicates how important the data is to the completion of the mission.

Spectral environment: A benchmark test will define the spectral environment, which can be specified based on (i) whether or not it is open, congested, or contested, (ii) how much interference is created by noncooperative nodes and on what frequencies, (iii) and are the noncooperative nodes passive or adaptive

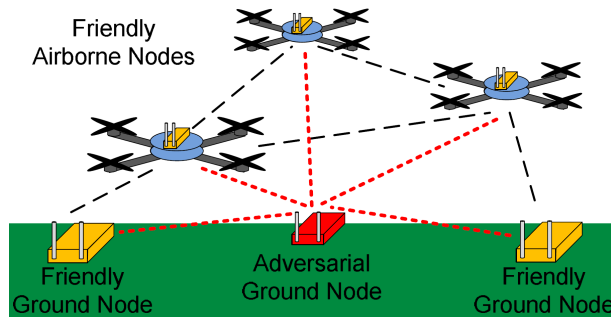


Figure 6: Network on UB-ANC agents

in how they create interference or jam communications.

Metrics: A benchmark test defines appropriate metrics that can be used to evaluate how well the underlying communications and networking capabilities support the mission’s completion. Data collected will include cyber metrics such as throughput, delay, and packet losses for different traffic priority classes, and physical metrics such as time-to-completion of individual tasks, time-to-completion of the overall mission, distance traveled per UAV, as well as GPS locations, velocities, and altitudes of the UAVs over time.

After defining various benchmark test scenarios, we will be able rigorously and systematically compare different airborne networking and communications protocols as well as command and control strategies in real network environments.

UB-ANC Behavioral Emulator: Another road map of the project is to provide an emulator for the drone. As we discussed before, the Agent software component determines the behavior of the drone in the network. We are currently designing a simulation software called the UB-ANC Behavioral Emulator, which will emulate the agent software component, mainly the `UBAgent` class, and visualize the result in a 3D rendering engine. This will allow us to emulate and verify the drones behaviors before they are deployed in a real environment.

Network and Cyber-Physical System Optimization: Lastly, we plan to deploy our air-

borne nodes within two network optimization frameworks, namely, ROSA (ROuting and Spectrum Allocation [50]) and ROCH (joint Routing and cOde-waveform CHannelization [63]), which have already been implemented and tested on (non-airborne) USRPs using GNU Radio. In addition, we plan to augment these network optimization frameworks to consider the cyber-physical aspects of the system; that is, select the (x, y, z) -coordinates in space where we can place the airborne nodes to maximize network throughput.

7 Conclusions

In this paper, we presented the design and implementation of the University at Buffalo’s Software-Defined Airborne Networking and Communications Testbed, which we call UB-ANC. To the best of our knowledge, UB-ANC is the first implemented airborne networking and communications testbed that includes (i) quadcopters capable of autonomous flight equipped with highly flexible and reconfigurable SDR-based transceivers (as opposed to only being equipped with SDR-based receivers or inflexible off-the-shelf transceivers, e.g., Wi-Fi or Zigbee); (ii) supports fully distributed command and control by endowing every airborne node with the ability to send commands to other nodes; (iii) is based entirely on open software and hardware; and (iv) aims to facilitate rapid testing and repeatable comparative evaluation of various airborne networking and communications protocols at different layers of the network protocol stack in a variety of benchmark test scenarios. To wrap up, we bring power and flexibility to the drone networks by combining three open platform components, USRP E310, GNU Radio, and C++ Qt.

8 Acknowledgments

We would like to gratefully acknowledge the help of Prince Joseph, who carefully designed our custom-built quadcopters, and Ian Hague, who provided expertise in flying the quadcopters and helped us measure the achievable flight times.

ACKNOWLEDGEMENT OF SUPPORT AND DISCLAIMER: (a) The University at Buffalo acknowledges the U.S. Government's support in the publication of this paper. This material is based upon work funded by the US Air Force Research Laboratory under Grant No. FA8750-14-1-0073. (b) Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL.

9 Availability

Git [14] is used for version control of the UB-ANC software, which is available via GitHub server. We provide step-by-step instructions on how to do things like porting Qt5, QtSerialPort, etc.

<https://github.com/jmodares/UB-ANC>

A more detailed explanation about the quadcopter platform, frame design, analysis, BOM, costs, assembly, safety checklist, etc. can be found in the docs folder.

References

- [1] 3D Robotics uBlox GPS with Compass Kit. <https://store.3drobotics.com/products/3dr-gps-ublox-with-compass>.
- [2] AD9361 RFIC, Analog Devices. <http://www.analog.com/en/products/rf-microwave/integrated-transceivers-transmitters-receivers/wideband-transceivers-ic/ad9361.html>.
- [3] Advanced eXtensible Interface, AMBA, ARM. <http://www.arm.com/products/system-ip/amba-specifications.php>.
- [4] ALGLIB Advance Library. <http://www.alglib.net>.
- [5] APM 2.6, 3D Robotics. <https://store.3drobotics.com/products/apm-2-dot-6-plus-assembled-set-side-entry>.
- [6] APM Planner 2, DIYDrones. https://github.com/diydrones/apm_planner.
- [7] APM:Copter, ArduPilot. <http://copter.ardupilot.com>.
- [8] Arduino, Open Platform. <https://www.arduino.cc>.
- [9] Ardupilot Subset MAVLink Commands. http://planner.ardupilot.com/wiki/common-mavlink-mission-command-messages-mav_cmd.
- [10] Artix-7 FPGA, Xilinx. <http://www.xilinx.com/products/silicon-devices/fpga/artix-7.html>.
- [11] ATmega2560, 8-bit AVR RISC-Based Microcontroller, ATMEL. <http://www.atmel.com/devices/atmega2560.aspx>.
- [12] Core concepts of GNU Radio. <https://gnuradio.org/redmine/projects/gnuradio/wiki/TutorialsCoreConcepts>.
- [13] Factor of Safety and Margin of Safety. <http://www.mechanical360.net/updates/factor-of-safety-and-margin-of-safety>.
- [14] Git, Distributed Version Control System. <https://git-scm.com>.
- [15] GNU Radio Companion. <https://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion>.
- [16] GNU Radio Project. <http://www.gnuradio.org>.
- [17] HEARTBEAT Message, MAVLink. <https://pixhawk.ethz.ch/mavlink/#HEARTBEAT>.
- [18] Information Directorate: Air Force Research Laboratory Rome Research Site. http://cybernyalliance.org/wp-content/uploads/2012/09/RI-Technical-Brochure_sm.pdf.
- [19] Inter Process Communication. <http://www.cs.cmu.edu/~IPC>.
- [20] Matchstiq, Epiq Solutions. <http://epiqsolutions.com/matchstiq>.
- [21] Matrice 100, DJI. <http://store.dji.com/product/matrice-100>.
- [22] MAVLink Commands Reference. <https://pixhawk.ethz.ch/mavlink>.
- [23] MAVLink Generator (C/C++, Python). <http://qgroundcontrol.org/mavlink/generator>.
- [24] MAVLink Ground Control Integration. http://qgroundcontrol.org/dev/mavlink_groundcontrol_integration_tutorial.
- [25] MAVLink Protocol. <http://qgroundcontrol.org/mavlink/start>.
- [26] Multi-Agent Distributed Adaptive Resource Allocation. <http://madara.sourceforge.net>.
- [27] NEON, ARM. <http://www.arm.com/products/processors/technologies/neon.php>.
- [28] Open Source Micro Air Vehicle Ground Control Station. <http://qgroundcontrol.org>.
- [29] Out-of-Tree Modules, GNU Radio. <https://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules>.
- [30] Pixhawk, 3D Robotics. <https://store.3drobotics.com/products/3dr-pixhawk>.
- [31] Protocol Data Unit. http://www.its.bldrdoc.gov/fs-1037/dir-028/_4199.htm.

- [32] PX4 Autopilot Project. <https://pixhawk.org>.
- [33] QObject Class, Qt5. <http://doc.qt.io/qt-5/qobject.html>.
- [34] QsLog Logging Library. <https://bitbucket.org/razvanpetru/qt-components/wiki/QsLog>.
- [35] Qt Application Framework. <http://www.qt.io>.
- [36] QThread Class, Qt5. <http://doc.qt.io/qt-5/qthread.html>.
- [37] Received Signal Strength Indicator. <https://supportforums.cisco.com/document/49506/snr-rssi-eirp-and-free-space-path-loss>.
- [38] RF Network on Chip, Ettus Research. <http://www.ettus.com/sdr-software/detail/rf-network-on-chip>.
- [39] Turnigy Talon Quadcopter V2.0 Carbon Fiber Frame. http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=67787.
- [40] USB CDC-ACM, Linux Kernel. <https://www.kernel.org/doc/Documentation/usb/acm.txt>.
- [41] USRP E310, Ettus Research. <http://www.ettus.com/product/details/E310-KIT>.
- [42] USRP E310 Schematics, Ettus Research. <http://files.ettus.com/schematics/e310>.
- [43] USRP Hardware Driver, Ettus Research. <http://files.ettus.com/manual>.
- [44] Vector-Optimized Library of Kernels. <http://libvolk.org>.
- [45] Yocto Project, an Embedded Linux Distro. <https://www.yoctoproject.org>.
- [46] Zynq-7000 All Programmable SoC, Xilinx. <http://www.xilinx.com/zynq>.
- [47] ALLRED, J., HASAN, A. B., PANICHSAKUL, S., PISANO, W., GRAY, P., HUANG, J., HAN, R., LAWRENCE, D., AND MOHSENI, K. Sensorflock: an airborne wireless sensor network of micro-air vehicles. In *Proceedings of the 5th international conference on Embedded networked sensor systems* (2007), ACM, pp. 117–129.
- [48] CHENG, B.-N., CHARLAND, R., CHRISTENSEN, P., COYLE, A., KUCZYNSKI, E., MCGARRY, S., PEDAN, I., VEYTSER, L., AND WHEELER, J. Characterizing routing with radio-router information in an airborne network. In *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011* (2011), IEEE, pp. 1985–1990.
- [49] CHIANG, M., LOW, S. H., DOYLE, J. C., ET AL. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE* 95, 1 (2007), 255–312.
- [50] DING, L., MELODIA, T., BATALAMA, S. N., MATYJAS, J. D., AND MEDLEY, M. J. Cross-layer routing and dynamic spectrum allocation in cognitive radio ad hoc networks. *Vehicular Technology, IEEE Transactions on* 59, 4 (2010), 1969–1979.
- [51] DOS SANTOS, G. A. M., BARNES, Z., LO, E., RITOPER, B., NISHIZAKI, L., TEJEDA, X., KE, A., LIN, H., SCHURGERS, C., LIN, A., ET AL. Small unmanned aerial vehicle system for wildlife radio collar tracking. In *Mobile Ad Hoc and Sensor Systems (MASS), 2014 IEEE 11th International Conference on* (2014), IEEE, pp. 761–766.
- [52] EDMONDSON, J., CAHILL, G., AND ROWE, A. On developing user interfaces for piloting unmanned systems. Available online: <http://www.cse.buffalo.edu/faculty/kdantu/rsn14/papers/edmondson.pdf>.
- [53] FREW, E. W., AND BROWN, T. X. Airborne communication networks for small unmanned aircraft systems. *Proceedings of the IEEE* 96, 12 (2008).
- [54] GEORGIADIS, L., NEELY, M. J., AND TASSIULAS, L. *Resource allocation and cross-layer control in wireless networks*. Now Publishers Inc, 2006.
- [55] JAKUBIAK, M. Cellular network coverage analysis using uav and sdr.
- [56] KUSHLEYEV, A., MELLINGER, D., POWERS, C., AND KUMAR, V. Towards a swarm of agile micro quadrotors. *Autonomous Robots* 35, 4 (2013), 287–300.
- [57] LE, T., KUTHETHOOR, G., HANSUPICHON, C., SESA, P., STROHM, J., HADYNSKI, G., KIWIOR, D., AND PARKER, D. Reliable user datagram protocol for airborne network. In *Military Communications Conference, 2009. MILCOM 2009. IEEE* (2009), IEEE, pp. 1–6.
- [58] NAMUDURI, K., WAN, Y., GOMATHISANKARAN, M., AND PENDSE, R. Airborne network: a cyber-physical system perspective. In *Proceedings of the first ACM MobiHoc workshop on Airborne Networks and Communications* (2012), ACM, pp. 55–60.
- [59] NYCHIS, G., HOTTELIER, T., YANG, Z., SESHAN, S., AND STEENKISTE, P. Enabling mac protocol implementations on software-defined radios. In *NSDI* (2009), vol. 9, pp. 91–105.
- [60] ROHRER, J. P., JABBAR, A., CETINKAYA, E. K., PERRINS, E., AND STERBENZ, J. P. Highly-dynamic cross-layered aeronautical network architecture. *Aerospace and Electronic Systems, IEEE Transactions on* 47, 4 (2011), 2742–2765.
- [61] SHIANG, H.-P., AND VAN DER SCHAAR, M. Online learning in autonomic multi-hop wireless networks for transmitting mission-critical applications. *Selected Areas in Communications, IEEE Journal on* 28, 5 (2010), 728–741.
- [62] SKLIVANITIS, G., DEMIRORS, E., BATALAMA, S., PADOS, D., MELODIA, T., AND MATYJAS, J. D. Demonstration of all-spectrum cognitive channelization on gnu radio and usrp-n210. In *Proc. of the NATO Symp. on "Cognitive Radio and Future Networks" (IST-123), The Netherlands* (2014).

- [63] SKLIVANITIS, G., DEMIRORS, E., BATALAMA, S. N., PADOS, D. A., AND MELODIA, T. Demo: Roch: Software-defined radio toolbox for experimental evaluation of all-spectrum cognitive networking. In *Proceedings of the 2015 Workshop on Software Radio Implementation Forum* (2015), ACM, pp. 10–10.
- [64] TIWARI, A., GANGULI, A., AND SAMPATH, A. Towards a mission planning toolbox for the airborne network: Optimizing ground coverage under connectivity constraints. In *Aerospace Conference, 2008 IEEE* (2008), IEEE, pp. 1–9.
- [65] TIWARI, A., GANGULI, A., SAMPATH, A., ANDERSON, D. S., SHEN, B.-H., KRISHNAMURTHI, N., YADEGAR, J., GERLA, M., AND KRZYSIAK, D. Mobility aware routing for the airborne network backbone. In *Military Communications Conference, 2008. MILCOM 2008. IEEE* (2008), IEEE, pp. 1–7.
- [66] ZHANG, P. Sensorfly: a minimalist approach for emergency situational awareness.
- [67] ZHOU, Y. Future communication model for high-speed railway based on unmanned aerial vehicles. *arXiv preprint arXiv:1411.3450* (2014).